NASA-IVV-97-008

NASA IV&V Facility, Fairmont, West Virginia

A Taxonomy of Object-Oriented Measures

Modeling the Object-Oriented Space

Ralph D. Neal, H. Roland Weistroffer, and Richard J. Coppins

July 14, 1997

This technical report is a product of the National Aeronautics and Space Administration (NASA) Software Program, an agency wide program to promote continual improvement of software engineering within NASA. The goals and strategies of this program are documented in the NASA software strategic plan, July 13, 1995.

Additional information is available from the NASA Software IV&V Facility on the World Wide Web site http://www.ivv.nasa.gov/

This research was funded under cooperative Agreement #NCC 2-979 at the NASA/WVU Software Research Laboratory.

A Taxonomy of Object-Oriented Measures
Modeling the Object-Oriented Space

Ralph D. Neal
NASA/WVU Software Research Laboratory, West Virginia University, 100 University Drive, Fairmont, WV 26554, U.S.A.
Email: rneal@research.ivv.nasa.gov

H. Roland Weistroffer
School of Business, Virginia Commonwealth University, PO Box 844000, Richmond, VA 23284-4000
Email: hrweistr@vcu.edu

Richard J. Coppins
School of Business, Virginia Commonwealth University, PO Box 844000, Richmond, VA 23284-4000
Email: rcoppins@busnet.bus.vcu.edu

Abstract—In order to control the quality of software and the software development process, it is important to understand the measurement of software. A first step toward a better comprehension of software measurement is the categorization of software measures by some meaningful taxonomy. The most worthwhile taxonomy would capture the fundamental nature of the object-oriented (O-O) space. The principal characteristics of object-oriented software offer a starting point for such a categorization of measures. This paper introduces a taxonomy of measures based upon fourteen characteristics of object-oriented software gathered from the literature. This taxonomy allows us to easily see gaps or redundancies in the existing O-O measures. The taxonomy also clearly differentiates among taxa so that there is no ambiguity as to the taxon to which a measure belongs. The taxonomy has been populated with measures taken from the literature.[1]

INTRODUCTION

Software development historically has been the arena of the artist. Artistically crafted code often resulted in arcane algorithms or spaghetti code that was unintelligible to those who had to perform maintenance. Initially only very primitive measurements such as lines of code (LOC) and development time per stage of the development life cycle were collected. Projects often exceeded estimated time and budget. In the pursuit of greater productivity, software development evolved into software engineering. Part of the software engineering concept is the idea that the

[1] Funded in part by NASA Cooperative Agreement NCCW-0040

product should be controllable. Control of a process or product requires that the process or product is measurable; therefore, control of software requires software measures [3].

Measurement is the process whereby numbers or symbols are assigned to dimensions of entities in such a manner as to describe the dimension in a meaningful way [7]. An entity may be a thing or an event, i.e., a person, a play, a developed program or the development process. A dimension is a trait of the entity, such as the height of a person, the cost of a play, or the length of the development process. Obviously, the entity and the dimension to be measured must be specified in advance. We cannot take measurements and then apply them to just any dimensions. Unfortunately this is exactly what the software development community has been doing [8], e.g., lines-of-code, being a valid measure of size, has been used to "measure" the complexity of programs [19]. In order to truly understand software and the software development process, software measurement must be better understood. What are the dimensions that define software and how do we measure them? A beginning step toward understanding software measurement is the categorization of the measurements by some meaningful taxonomy.

Software and software development are extremely complex. We should not expect to be able to measure entities of such complexity with one, two, or even a dozen measures. Measures have to be developed to allow us to view software from many perspectives. In this paper, we differentiate between measures and metrics: A metric is defined to be any proposed type of measurement, not necessarily validated, whereas a measure must be validated. Thus all measures are metrics, but not all metrics are measures. Many object-oriented (O-O) metrics have been proposed in the literature, e.g., [2], [4], [5], [10], [11], and [12]. To better comprehend the contributions of these metrics, it is necessary to categorize them in a meaningful way so that the various dimensions of O-O software being measured can be better understood. The authors are not aware of any proposed organization of software measures or metrics in the published literature that models the object-oriented space in a comprehensive manner. Until we better understand the many dimensions of O-O software, we cannot truly understand the product. It does little good to measure the process if the product is not measured. Being the best at producing an inferior product does not define a quality process. To facilitate understanding of the product, this paper proposes a taxonomy that helps us model the object-oriented space and allows us to classify measures accordingly.

BACKGROUND

There has been little agreement among authors as to the characteristics that identify the object-oriented approach. Henderson-Sellers [9] lists information hiding, encapsulation, objects, classification, classes, abstraction, inheritance, polymorphism, dynamic binding, persistence, and composition as having been chosen by at least one author as a defining aspect of object-orientation. Rumbaugh, et al. [15] add identity, Smith [16] adds single type and Sully [17] adds the unit building block to this list of defining aspects. These characteristics of object-orientation are not completely disjoint: there is much overlapping of aspects as different authors group sub-aspects differently and create their own individual groupings, each with a unique aspect name. It should be clear from the preceding list that there are many dimensions to O-O software. It should also be noted that this list may not be exhaustive.

Archer and Stinson [2] propose a taxonomy that places a metric in one (or more) of five taxa, viz., system, coupling and uses, inheritance, class, and method. It is unclear where a measure of say coupling among methods, as in Tegarden, et al., [18], would be classified in this taxonomy. The coarseness of this taxonomy also causes metrics for different software artifacts to be grouped together, e.g., if all coupling metrics are classified as "coupling and uses" metrics, then system measures could be lumped together with measures of objects, measures of methods, and measures of variables (see system messages, object fan-in, method fan-in, and variable fan-in in Tegarden, et al., [18]). A useful taxonomy of software measures should clearly differentiate among taxa so that a measure belongs to one and only one taxon and there is no ambiguity as to the taxon to which a measure belongs.

The Tegarden, et al., [18] model of object-oriented systems complexity measurement defines object-oriented systems as looking different from different viewpoints. The model defines four levels of software strata that a software developer might want to measure (in order of granularity): variables, methods, objects, and systems. The model then defines characteristics (dimensions) that contribute to the character of each level. The model clearly differentiates among the four levels but is not so clear in defining the dimensions. For example, class design encompasses encapsulation, complexity, and polymorphism.

Abreu and Carapuca [1] also see the advantages of separating measures by level. However, the Abreu and Carapuca model also groups measures into large ambiguous lots. This model defines software entities (granularity) as method, class, and system, and it defines the dimensions within each level of granularity as design, size, complexity, reuse, productivity, and quality. This three by six model is less encompassing than the model of Tegarden, et al. Another problem with the Abreu and Carapuca model is the grouping of process measures, with product measures, e.g., productivity with size. While both process and product measures are useful, mixing them in the same taxonomy only clouds the two separate issues.

Beyond these models, little work on software measures taxonomies has been reported in the published literature. Building on the models by Tegarden, et al., [18] and Abreu and Carapuca [1], a new object-oriented measures taxonomy (see Table 1) is proposed. This taxonomy partitions the object-oriented space into non-overlapping domains so that measures of these domains can be fit unambiguously into one and only one taxon.

THE OBJECT-ORIENTED SPACE AND THE NEW TAXONOMY

In order to measure object-oriented software the measurer needs to be aware of the characteristics of O-O software and of the different levels of granularity inherent in the O-O paradigm. The Tegarden, et al., [18] model of object-oriented systems complexity measurement defines four levels of software strata. Neal [12] adds a fifth stratum to this. Building on this model, the object-oriented space can be represented as a matrix that partitions the space into several levels of granularity with multiple characteristics for each of these levels. The software levels that a software developer might want to measure (in order of granularity) are variables, methods, classes, programs, and systems. The levels are represented by the columns of the matrix. Each level of granularity exhibits characteristics (dimensions) that contribute to the character of that level. The dimensions are represented by the rows of the matrix. Fourteen dimensions of O-O software have been gleaned from the literature. This model partitions the object-oriented space into understandable, unambiguous segments, and thereby forces a reasonable structure upon measurers.

As has been stated earlier, a first step toward understanding software measurement better is the categorization of the measurements by some meaningful taxonomy. If we are to learn about the object-oriented space, it must be possible for diversified measurers to reach similar conclusions given the same data. A taxonomy should at the least allow each measurer to start from a common basis.

> **Postulate:** A useful software measures taxonomy should clearly differentiate among taxa so that there is no ambiguity as to the taxon to which a measure belongs.

Table 1 supplies a summary of measures or metrics for each of the fourteen dimensions across the five levels of granularity (70 cells). Though many of these fourteen dimensions appear repeatedly in the literature, they may not necessarily be the dimensions that matter the most. It is possible that there are other dimensions that do not yet have metrics proposed to measure them, but the measurement of which would be useful or necessary to understand an object-oriented artifact. Certainly not all fourteen dimensions apply to all levels, e.g., encapsulation does not apply to variable or method. The same dimension measured on different levels will

almost certainly require different measures or at least a different scope, e.g., lines-of-code (LOC) in a program vs. LOC in a system.

The object-oriented space matrix offers a starting point for such a categorization of measures. By filling in the cells of the object-oriented space matrix with the measures or metrics proposed in the literature, the matrix becomes the Object-Oriented Measures Taxonomy. This taxonomy includes all of the published, interesting characteristics of software and clearly defines where any measure fits among the taxa without worry of overlap or ambiguity. If a measure cannot be placed easily into one and only one taxon, the measure may not be well understood. A measure that is not well understood is unlikely to be useful to the measurer and should be discarded. It is also possible that if a measure cannot be placed easily into any existing taxon that the taxonomy is incomplete. In that case, more research may be needed to expand the taxonomy.

Table 1 has been populated with thirty measures taken from the literature [4,5,10,11,14]. These measures have been validated in the narrow sense of Fenton [7] using measurement theory with Zuse's augmentation [12, 13, 19, 20]. Every measure that could be validated in the narrow sense could also be categorized in this taxonomy. In addition to these validated measures, several unvalidated metrics from Tegarden, et al., [18] have been included in Table 1. They have been included to show that work is being done at the variable and method levels. These metrics in no way represent all of the metrics offered by Tegarden, et al. Additional metrics were not included for those cells for which validated measures already exist.

CONCLUSION AND FUTURE RESEARCH

As Table 1 shows, often there are multiple metrics available which attempt to measure the same dimension of the same level. The collection of measurement data is usually very expensive [6]; nevertheless, the application of multiple measures to measure the same dimension of the same level of software can be useful. The collection of data for multiple measures allows the measures to be compared to each other to either confirm that they do indeed measure the same dimension or establish that one (or more) of them is measuring something other than the dimension in question. Once it is established which measure most cost effectively measures the dimension in question, it may no longer be necessary to collect data for the other measures. If the measures in one cell are not all measuring the same dimension, then one or more of the measures may have been miscataloged.

As stated earlier, the fourteen dimensions used in the proposed taxonomy are those found in the literature. In other words, these are the dimensions that have been thought by the O-O community to be important enough to measure. Other dimensions that may be of equal or higher importance may yet be discovered. Because not all of the fourteen dimensions are applicable to all five levels some

cells in Table 1 should remain empty.

Some measures may be scaleable to levels other than that level for which they were designed. Measures that are scaleable are not directly applicable as defined but may lend themselves to being averaged or summed to fill a cell at a higher level. No measures have been found to be scaleable to cells at a lower level.

Much work remains before the nature of the object-oriented software development process can be sufficiently understood through the measurement of software products. More measures need to be developed to allow us to view software from its many perspectives, i.e., validated measures need to be found to fill more of the empty cells of the object-oriented measures taxonomy (Table 1). Further, the measures populating the proposed taxonomy need to be tested empirically. Software product measures ultimately are only useful when they can be shown to be reliable prediction variables of software development cost and schedule, software maintenance cost and schedule [7], or software performance.

If the product measure is to be used as a performance predictor, performance measures need to be established that baseline acceptable performance and act as outside variables against which to test the product measure. If performance measures cannot be established, it may be that this cell, i.e., this dimension at this level, is not important to the performance prediction system. If the product measure is to be used as a cost and schedule predictor, cost and schedule measures against which to test the product measure need to be collected (or calculated). If cost and schedule measures cannot be established, it may be that this cell is not important to the cost and schedule prediction system.

Product measures that are found to be too costly to collect need to be discarded. Likewise, product measures that are found to be ineffectual in the prediction systems also need to be eliminated. However, if performance or schedule and cost measures have been developed, other predictive measures must already exist or must be developed to fill the appropriate cell in the taxonomy. When product measures are eliminated, and the removal of the measure causes the cell to become empty, then other product measures may need to be developed to fill the void.

The taxonomy itself needs to be tested empirically. If meaningful measures cannot be defined for a specific cell (a given dimension at a given level), e.g., method encapsulation, then perhaps the cell should be blackened out. Likewise, if useful outside variables (performance, schedule, or cost) cannot be defined against which to test the measures of a cell then, again, perhaps the cell should be blackened out. If all levels of a dimension have been blackened out, the entire row (dimension) should be reviewed for possible removal from the taxonomy matrix. On the other hand, if a new dimension becomes apparent, a new row should be added to the taxonomy matrix. The new dimension then needs to be populated with validated measures.

Though extremely complex, software should be as measurable as any other complex entity, say, automobiles. If current wisdom holds, encapsulation may prove to be as important to the stability of object-oriented software as wheelbase is to the stability of an automobile.

Table 1 **Object-Oriented Measures Taxonomy**

Level⇒ Dimension⇓	Variable	Method	Class	Program	System
Clarity			CLM	CLM	CLM
Cohesion	*(1/ (vfi+vfo+vp))*	*(local (mfi+mfo) / total (mfi+mfo))*	**DMC** **DCWO**	**DMC** **DCWO**	**DMC** **DCWO**
Coupling	*(1-(1/ (vfi+vfo+vp)))*	*(remote(mfi+mfo) / total (mfi+mfo))*	**UCGU** **DCBO**	**UCGU** **DCBO**	**UCGU** **DCBO**
Complexity, inter-structural	*remote vfi* *remote vfo*	*remote mfi* *remote mfo* *remote I/Ov*	NIM PIM RFC	AIM PIM RFC	AIM PIM RFC
Complexity, intra-structural	*local vfi* *local vfo*	**SML** *local mfi* *local mfo* *local I/Ov*			
Complexity, psychological		**MAA** *I/Ov*	MPC WMC	MPC WMC	MPC WMC
Design			PRC NOM	PRC FOC	PRC FOC
Encapsulation			FFU	FFU	FFU
Inheritance	*vfd*	*mfd*	**PMI** **PMIS**	**DAC** NAC	**DAC** NAC
Information hiding			**PrIM**	**PrIM**	**PrIM**
Polymorphism	*vp*	*mp*	*(vp+mp) normal-ized*		
Reuse	*vfi-1*	*mfi-1*	RUS CRE	ΣRUS ΣCRE	ΣRUS ΣCRE
Size		LOC AMS	LOC AMS NIV	LOC AMS AIV	LOC AMS AIV
Specialization			**POM** NCM NMA	**POM** NCM NMA	**POM** NCM NMA

Measures from [12]

Measures from [4], [5], [10], and [11]

Metrics from [18]

Measures that can be scaled up to a higher level or derived from scales at a lower level

Note: The abbreviated names of the measures and metrics are explained in the Appendix.

APPENDIX: Definitions of measures and metrics

AIM Average number of instance methods per class [11]
AIV Average number of instance variables [11]
AMS Average method size [11]
CRE Number of times a class is reused [11]
CLM Average number of comment lines per method [11]
DAC Density of abstract classes [12]
DCBO Degree of coupling between classes [12]
DCWO Degree of coupling within classes [12]
DMC Density of methodological cohesiveness [12]
FFU Use of friend functions [11]
FOC Percentage of function-oriented code [11]
I/Ov *Input/output variables [18]*
LOC Lines of code [11]: Number of statements (NOS) [11]: Number of semicolons in a class (SIZE1) [10]
MAA Messages and arguments [12]
mfd *Method fan down [18]*
mfi *Method fan in [18]*
mfo *Method fan out [18]*
mp *Method polymorphism [18]*
MPC Message-passing coupling [10]
NAC Number of abstract classes [11]
NCM Number of class methods in a class [11]
NIM Number of instance methods in a class [11]
NIV Number of instance variables in a class [11]
NMA Number of methods added by a subclass [11]
NOM Number of local methods [10]
PIM Number of public instance methods in a class [11]
PMI Potential methods inherited [12]
PMIS Proportion of methods inherited by a subclass [12]
POM Proportion of overriding methods in a subclass [12]
PRC Number of problem reports per class or contract [11]
PrIM Number of private instance methods [12]
RFC Response for a class [5]
RUS Reuse of a class [4]
SML Strings of message-links [12]
ΣCRE Summation of CRE for all classes
ΣRUS Summation of RUS for all classes
UCGU Unnecessary coupling through global usage [12]
vfd *Variable fan down [18]*
vfi *Variable fan in [18]*
vfo *Variable fan out [18]*
vp *Variable polymorphism [18]*
WMC Weighted methods per class [5]

References

1. Abreu, Fernando Brito e, and Rogerio Carapuca, Candidate for Object-Oriented Software within a Taxonomy Framework, *Journal of Systems Software*, 1995, 26, 87-96.

2. Archer, Clark, and Michael Stinson, Object-Oriented Software Measures, *Technical Report CMU/SEI-95-TR-002*, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, 1995.

3. Baker, Albert L., James M. Bieman, Norman Fenton, Davis A. Gustafson, Austin Melton, and Robin Whitty, "A Philosophy of Software Measurement", *The Journal of Systems and Software*, Vol. 12, 1990, p. 277-281.

4. Chen, J-Y, and J-F Lu, A New Metric for Object-Oriented Design, *Information and Software Technology*, 1993, 232-240.

5. Chidamber, Shyam R., and Chris F. Kemerer, A Metric Suite for Object Oriented Design, *IEEE Transactions on Software Engineering*, 20:6, June 1994.

6. Deutsch, Michael S., and Ronald R. Willis, *Software Quality Engineering: A Total Technical and Management Approach*, Prentice-Hall, Englewood Cliffs, NJ, 1988.

7. Fenton, Norman, *Software Metrics: A Rigorous Approach*, Chapman & Hall, London, UK, 1991.

8. Fenton, Norman, Software Measurement: A Necessary Scientific Basis, *IEEE Transactions on Software Engineering*, 20:3, March 1994.

9. Henderson-Sellers, B., *A Book of Object-Oriented Knowledge*, Prentice Hall, NY, 1992.

10. Li, Wei, and Sallie Henry, Maintenance Metrics for the Object-Oriented Paradigm, *Proceedings of the First International Software Metrics Symposium*, May 1993b.

11. Lorenz, Mark, and Jeff Kidd, *Object-Oriented Software Metrics*, Prentice Hall, Englewood Cliffs, NJ, 1994.

12. Neal, Ralph D., *The Validation by Measurement Theory of Proposed Object-Oriented Software Metrics*, Dissertation, School of Business, Virginia Commonwealth University, Richmond, VA, 1996.

13. Neal, R.D., R.J. Coppins, and H.R. Weistroffer, *The Assignment of Scale to Object-Oriented Software Measures*, Working Paper, Virginia Commonwealth University, Richmond, VA, 1997.

14. Neal, R.D., H.R. Weistroffer, and R.J. Coppins, *An Improved Suite of Object-Oriented Software Measures*, Working Paper, Virginia Commonwealth University, Richmond, VA, 1997.

15. Rumbaugh, James, Michael Blaha, William Premerlani, Frederick Eddy, and William Lorensen, *Object-Oriented Modeling and Design*, Prentice Hall, Englewood Cliffs, NJ, 1991.

16. Smith, David N., *Concepts of Object-Oriented Programming*, McGraw-Hill, NY, 1991.

17. Sully, Phil, *Modeling the World with Objects*, Prentice Hall, NY, 1993.

18. Tegarden, David P., Steven D. Sheetz, and David E. Monarchi, A Software Complexity Model of Object-Oriented Systems, *Decision Support Systems 13*, 1995, 241-62.

19. Zuse, Horst, *Software Complexity: Measures and Methods*, Walter de Gruyter, Berlin, 1990.

20. Zuse, Horst, "Foundations of Object-Oriented Software Measures", *Proceedings of the Third International Software Metrics Symposium*, March 1996.

A Taxonomy of Object-Oriented Measures
Modeling the Object-Oriented Space

Ralph D. Neal
NASA/WVU Software Research Laboratory, West Virginia University, 100 University Drive, Fairmont, WV 26554, U.S.A.
Email: rneal@research.ivv.nasa.gov

H. Roland Weistroffer
School of Business, Virginia Commonwealth University, PO Box 844000, Richmond, VA 23284-4000
Email: hrweistr@vcu.edu

Richard J. Coppins
School of Business, Virginia Commonwealth University, PO Box 844000, Richmond, VA 23284-4000
Email: rcoppins@busnet.bus.vcu.edu

Abstract—In order to control the quality of software and the software development process, it is important to understand the measurement of software. A first step toward a better comprehension of software measurement is the categorization of software measures by some meaningful taxonomy. The most worthwhile taxonomy would capture the fundamental nature of the object-oriented (O-O) space. The principal characteristics of object-oriented software offer a starting point for such a categorization of measures. This paper introduces a taxonomy of measures based upon fourteen characteristics of object-oriented software gathered from the literature. This taxonomy allows us to easily see gaps or redundancies in the existing O-O measures. The taxonomy also clearly differentiates among taxa so that there is no ambiguity as to the taxon to which a measure belongs. The taxonomy has been populated with measures taken from the literature.[1]

INTRODUCTION

Software development historically has been the arena of the artist. Artistically crafted code often resulted in arcane algorithms or spaghetti code that was unintelligible to those who had to perform maintenance. Initially only very primitive measurements such as lines of code (LOC) and development time per stage of the development life cycle were collected. Projects often exceeded estimated time and budget. In the pursuit of greater productivity, software development evolved into software engineering. Part of the software engineering concept is the idea that the

[1] Funded in part by NASA Cooperative Agreement NCCW-0040

product should be controllable. Control of a process or product requires that the process or product is measurable; therefore, control of software requires software measures [3].

Measurement is the process whereby numbers or symbols are assigned to dimensions of entities in such a manner as to describe the dimension in a meaningful way [7]. An entity may be a thing or an event, i.e., a person, a play, a developed program or the development process. A dimension is a trait of the entity, such as the height of a person, the cost of a play, or the length of the development process. Obviously, the entity and the dimension to be measured must be specified in advance. We cannot take measurements and then apply them to just any dimensions. Unfortunately this is exactly what the software development community has been doing [8], e.g., lines-of-code, being a valid measure of size, has been used to "measure" the complexity of programs [19]. In order to truly understand software and the software development process, software measurement must be better understood. What are the dimensions that define software and how do we measure them? A beginning step toward understanding software measurement is the categorization of the measurements by some meaningful taxonomy.

Software and software development are extremely complex. We should not expect to be able to measure entities of such complexity with one, two, or even a dozen measures. Measures have to be developed to allow us to view software from many perspectives. In this paper, we differentiate between measures and metrics: A metric is defined to be any proposed type of measurement, not necessarily validated, whereas a measure must be validated. Thus all measures are metrics, but not all metrics are measures. Many object-oriented (O-O) metrics have been proposed in the literature, e.g., [2], [4], [5], [10], [11], and [12]. To better comprehend the contributions of these metrics, it is necessary to categorize them in a meaningful way so that the various dimensions of O-O software being measured can be better understood. The authors are not aware of any proposed organization of software measures or metrics in the published literature that models the object-oriented space in a comprehensive manner. Until we better understand the many dimensions of O-O software, we cannot truly understand the product. It does little good to measure the process if the product is not measured. Being the best at producing an inferior product does not define a quality process. To facilitate understanding of the product, this paper proposes a taxonomy that helps us model the object-oriented space and allows us to classify measures accordingly.

BACKGROUND

There has been little agreement among authors as to the characteristics that identify the object-oriented approach. Henderson-Sellers [9] lists information hiding, encapsulation, objects, classification, classes, abstraction, inheritance, polymorphism, dynamic binding, persistence, and composition as having been chosen by at least one author as a defining aspect of object-orientation. Rumbaugh, et al. [15] add identity, Smith [16] adds single type and Sully [17] adds the unit building block to this list of defining aspects. These characteristics of object-orientation are not completely disjoint: there is much overlapping of aspects as different authors group sub-aspects differently and create their own individual groupings, each with a unique aspect name. It should be clear from the preceding list that there are many dimensions to O-O software. It should also be noted that this list may not be exhaustive.

Archer and Stinson [2] propose a taxonomy that places a metric in one (or more) of five taxa, viz., system, coupling and uses, inheritance, class, and method. It is unclear where a measure of say coupling among methods, as in Tegarden, et al., [18], would be classified in this taxonomy. The coarseness of this taxonomy also causes metrics for different software artifacts to be grouped together, e.g., if all coupling metrics are classified as "coupling and uses" metrics, then system measures could be lumped together with measures of objects, measures of methods, and measures of variables (see system messages, object fan-in, method fan-in, and variable fan-in in Tegarden, et al., [18]). A useful taxonomy of software measures should clearly differentiate among taxa so that a measure belongs to one and only one taxon and there is no ambiguity as to the taxon to which a measure belongs.

The Tegarden, et al., [18] model of object-oriented systems complexity measurement defines object-oriented systems as looking different from different viewpoints. The model defines four levels of software strata that a software developer might want to measure (in order of granularity): variables, methods, objects, and systems. The model then defines characteristics (dimensions) that contribute to the character of each level. The model clearly differentiates among the four levels but is not so clear in defining the dimensions. For example, class design encompasses encapsulation, complexity, and polymorphism.

Abreu and Carapuca [1] also see the advantages of separating measures by level. However, the Abreu and Carapuca model also groups measures into large ambiguous lots. This model defines software entities (granularity) as method, class, and system, and it defines the dimensions within each level of granularity as design, size, complexity, reuse, productivity, and quality. This three by six model is less encompassing than the model of Tegarden, et al. Another problem with the Abreu and Carapuca model is the grouping of process measures, with product measures, e.g., productivity with size. While both process and product measures are useful, mixing them in the same taxonomy only clouds the two separate issues.

Beyond these models, little work on software measures taxonomies has been reported in the published literature. Building on the models by Tegarden, et al., [18] and Abreu and Carapuca [1], a new object-oriented measures taxonomy (see Table 1) is proposed. This taxonomy partitions the object-oriented space into non-overlapping domains so that measures of these domains can be fit unambiguously into one and only one taxon.

THE OBJECT-ORIENTED SPACE AND THE NEW TAXONOMY

In order to measure object-oriented software the measurer needs to be aware of the characteristics of O-O software and of the different levels of granularity inherent in the O-O paradigm. The Tegarden, et al., [18] model of object-oriented systems complexity measurement defines four levels of software strata. Neal [12] adds a fifth stratum to this. Building on this model, the object-oriented space can be represented as a matrix that partitions the space into several levels of granularity with multiple characteristics for each of these levels. The software levels that a software developer might want to measure (in order of granularity) are variables, methods, classes, programs, and systems. The levels are represented by the columns of the matrix. Each level of granularity exhibits characteristics (dimensions) that contribute to the character of that level. The dimensions are represented by the rows of the matrix. Fourteen dimensions of O-O software have been gleaned from the literature. This model partitions the object-oriented space into understandable, unambiguous segments, and thereby forces a reasonable structure upon measurers.

As has been stated earlier, a first step toward understanding software measurement better is the categorization of the measurements by some meaningful taxonomy. If we are to learn about the object-oriented space, it must be possible for diversified measurers to reach similar conclusions given the same data. A taxonomy should at the least allow each measurer to start from a common basis.

> **Postulate:** A useful software measures taxonomy should clearly differentiate among taxa so that there is no ambiguity as to the taxon to which a measure belongs.

Table 1 supplies a summary of measures or metrics for each of the fourteen dimensions across the five levels of granularity (70 cells). Though many of these fourteen dimensions appear repeatedly in the literature, they may not necessarily be the dimensions that matter the most. It is possible that there are other dimensions that do not yet have metrics proposed to measure them, but the measurement of which would be useful or necessary to understand an object-oriented artifact. Certainly not all fourteen dimensions apply to all levels, e.g., encapsulation does not apply to variable or method. The same dimension measured on different levels will

almost certainly require different measures or at least a different scope, e.g., lines-of-code (LOC) in a program vs. LOC in a system.

The object-oriented space matrix offers a starting point for such a categorization of measures. By filling in the cells of the object-oriented space matrix with the measures or metrics proposed in the literature, the matrix becomes the Object-Oriented Measures Taxonomy. This taxonomy includes all of the published, interesting characteristics of software and clearly defines where any measure fits among the taxa without worry of overlap or ambiguity. If a measure cannot be placed easily into one and only one taxon, the measure may not be well understood. A measure that is not well understood is unlikely to be useful to the measurer and should be discarded. It is also possible that if a measure cannot be placed easily into any existing taxon that the taxonomy is incomplete. In that case, more research may be needed to expand the taxonomy.

Table 1 has been populated with thirty measures taken from the literature [4,5,10,11,14]. These measures have been validated in the narrow sense of Fenton [7] using measurement theory with Zuse's augmentation [12, 13, 19, 20]. Every measure that could be validated in the narrow sense could also be categorized in this taxonomy. In addition to these validated measures, several unvalidated metrics from Tegarden, et al., [18] have been included in Table 1. They have been included to show that work is being done at the variable and method levels. These metrics in no way represent all of the metrics offered by Tegarden, et al. Additional metrics were not included for those cells for which validated measures already exist.

CONCLUSION AND FUTURE RESEARCH

As Table 1 shows, often there are multiple metrics available which attempt to measure the same dimension of the same level. The collection of measurement data is usually very expensive [6]; nevertheless, the application of multiple measures to measure the same dimension of the same level of software can be useful. The collection of data for multiple measures allows the measures to be compared to each other to either confirm that they do indeed measure the same dimension or establish that one (or more) of them is measuring something other than the dimension in question. Once it is established which measure most cost effectively measures the dimension in question, it may no longer be necessary to collect data for the other measures. If the measures in one cell are not all measuring the same dimension, then one or more of the measures may have been miscataloged.

As stated earlier, the fourteen dimensions used in the proposed taxonomy are those found in the literature. In other words, these are the dimensions that have been thought by the O-O community to be important enough to measure. Other dimensions that may be of equal or higher importance may yet be discovered. Because not all of the fourteen dimensions are applicable to all five levels some

cells in Table 1 should remain empty.

Some measures may be scaleable to levels other than that level for which they were designed. Measures that are scaleable are not directly applicable as defined but may lend themselves to being averaged or summed to fill a cell at a higher level. No measures have been found to be scaleable to cells at a lower level.

Much work remains before the nature of the object-oriented software development process can be sufficiently understood through the measurement of software products. More measures need to be developed to allow us to view software from its many perspectives, i.e., validated measures need to be found to fill more of the empty cells of the object-oriented measures taxonomy (Table 1). Further, the measures populating the proposed taxonomy need to be tested empirically. Software product measures ultimately are only useful when they can be shown to be reliable prediction variables of software development cost and schedule, software maintenance cost and schedule [7], or software performance.

If the product measure is to be used as a performance predictor, performance measures need to be established that baseline acceptable performance and act as outside variables against which to test the product measure. If performance measures cannot be established, it may be that this cell, i.e., this dimension at this level, is not important to the performance prediction system. If the product measure is to be used as a cost and schedule predictor, cost and schedule measures against which to test the product measure need to be collected (or calculated). If cost and schedule measures cannot be established, it may be that this cell is not important to the cost and schedule prediction system.

Product measures that are found to be too costly to collect need to be discarded. Likewise, product measures that are found to be ineffectual in the prediction systems also need to be eliminated. However, if performance or schedule and cost measures have been developed, other predictive measures must already exist or must be developed to fill the appropriate cell in the taxonomy. When product measures are eliminated, and the removal of the measure causes the cell to become empty, then other product measures may need to be developed to fill the void.

The taxonomy itself needs to be tested empirically. If meaningful measures cannot be defined for a specific cell (a given dimension at a given level), e.g., method encapsulation, then perhaps the cell should be blackened out. Likewise, if useful outside variables (performance, schedule, or cost) cannot be defined against which to test the measures of a cell then, again, perhaps the cell should be blackened out. If all levels of a dimension have been blackened out, the entire row (dimension) should be reviewed for possible removal from the taxonomy matrix. On the other hand, if a new dimension becomes apparent, a new row should be added to the taxonomy matrix. The new dimension then needs to be populated with validated measures.

Though extremely complex, software should be as measurable as any other complex entity, say, automobiles. If current wisdom holds, encapsulation may prove to be as important to the stability of object-oriented software as wheelbase is to the stability of an automobile.

Table 1 **Object-Oriented Measures Taxonomy**

Level⇒ Dimension⇓	Variable	Method	Class	Program	System
Clarity			CLM	CLM	CLM
Cohesion	*(1/ (vfi+vfo+vp))*	*(local (mfi+mfo) / total (mfi+mfo))*	**DMC** **DCWO**	**DMC** **DCWO**	**DMC** **DCWO**
Coupling	*(1-(1/ (vfi+vfo+vp)))*	*(remote(mfi+mfo) / total (mfi+mfo))*	**UCGU** **DCBO**	**UCGU** **DCBO**	**UCGU** **DCBO**
Complexity, inter-structural	*remote vfi* *remote vfo*	*remote mfi* *remote mfo* *remote I/Ov*	NIM PIM RFC	AIM PIM RFC	AIM PIM RFC
Complexity, intra-structural	*local vfi* *local vfo*	**SML** *local mfi* *local mfo* *local I/Ov*			
Complexity, psychological		**MAA** *I/Ov*	MPC WMC	MPC WMC	MPC WMC
Design			PRC NOM	PRC FOC	PRC FOC
Encapsulation			FFU	FFU	FFU
Inheritance	*vfd*	*mfd*	**PMI** **PMIS**	**DAC** NAC	**DAC** NAC
Information hiding			**PrIM**	**PrIM**	**PrIM**
Polymorphism	*vp*	*mp*	*(vp+mp) normal-ized*		
Reuse	*vfi-1*	*mfi-1*	RUS CRE	ΣRUS ΣCRE	ΣRUS ΣCRE
Size		LOC AMS	LOC AMS NIV	LOC AMS AIV	LOC AMS AIV
Specialization			**POM** NCM NMA	**POM** NCM NMA	**POM** NCM NMA

Measures from [12]

Measures from [4], [5], [10], and [11]

Metrics from [18]

Measures that can be scaled up to a higher level or derived from scales at a lower level

Note: The abbreviated names of the measures and metrics are explained in the Appendix.

APPENDIX: Definitions of measures and metrics

AIM	Average number of instance methods per class [11]
AIV	Average number of instance variables [11]
AMS	Average method size [11]
CRE	Number of times a class is reused [11]
CLM	Average number of comment lines per method [11]
DAC	Density of abstract classes [12]
DCBO	Degree of coupling between classes [12]
DCWO	Degree of coupling within classes [12]
DMC	Density of methodological cohesiveness [12]
FFU	Use of friend functions [11]
FOC	Percentage of function-oriented code [11]
I/Ov	*Input/output variables [18]*
LOC	Lines of code [11]: Number of statements (NOS) [11]: Number of semicolons in a class (SIZE1) [10]
MAA	Messages and arguments [12]
mfd	*Method fan down [18]*
mfi	*Method fan in [18]*
mfo	*Method fan out [18]*
mp	*Method polymorphism [18]*
MPC	Message-passing coupling [10]
NAC	Number of abstract classes [11]
NCM	Number of class methods in a class [11]
NIM	Number of instance methods in a class [11]
NIV	Number of instance variables in a class [11]
NMA	Number of methods added by a subclass [11]
NOM	Number of local methods [10]
PIM	Number of public instance methods in a class [11]
PMI	Potential methods inherited [12]
PMIS	Proportion of methods inherited by a subclass [12]
POM	Proportion of overriding methods in a subclass [12]
PRC	Number of problem reports per class or contract [11]
PrIM	Number of private instance methods [12]
RFC	Response for a class [5]
RUS	Reuse of a class [4]
SML	Strings of message-links [12]
ΣCRE	Summation of CRE for all classes
ΣRUS	Summation of RUS for all classes
UCGU	Unnecessary coupling through global usage [12]
vfd	*Variable fan down [18]*
vfi	*Variable fan in [18]*
vfo	*Variable fan out [18]*
vp	*Variable polymorphism [18]*
WMC	Weighted methods per class [5]

References

1. Abreu, Fernando Brito e, and Rogerio Carapuca, Candidate for Object-Oriented Software within a Taxonomy Framework, *Journal of Systems Software*, 1995, 26, 87-96.

2. Archer, Clark, and Michael Stinson, Object-Oriented Software Measures, *Technical Report CMU/SEI-95-TR-002*, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, 1995.

3. Baker, Albert L., James M. Bieman, Norman Fenton, Davis A. Gustafson, Austin Melton, and Robin Whitty, "A Philosophy of Software Measurement", *The Journal of Systems and Software*, Vol. 12, 1990, p. 277-281.

4. Chen, J-Y, and J-F Lu, A New Metric for Object-Oriented Design, *Information and Software Technology*, 1993, 232-240.

5. Chidamber, Shyam R., and Chris F. Kemerer, A Metric Suite for Object Oriented Design, *IEEE Transactions on Software Engineering*, 20:6, June 1994.

6. Deutsch, Michael S., and Ronald R. Willis, *Software Quality Engineering: A Total Technical and Management Approach*, Prentice-Hall, Englewood Cliffs, NJ, 1988.

7. Fenton, Norman, *Software Metrics: A Rigorous Approach*, Chapman & Hall, London, UK, 1991.

8. Fenton, Norman, Software Measurement: A Necessary Scientific Basis, *IEEE Transactions on Software Engineering*, 20:3, March 1994.

9. Henderson-Sellers, B., *A Book of Object-Oriented Knowledge*, Prentice Hall, NY, 1992.

10. Li, Wei, and Sallie Henry, Maintenance Metrics for the Object-Oriented Paradigm, *Proceedings of the First International Software Metrics Symposium*, May 1993b.

11. Lorenz, Mark, and Jeff Kidd, *Object-Oriented Software Metrics*, Prentice Hall, Englewood Cliffs, NJ, 1994.

12. Neal, Ralph D., *The Validation by Measurement Theory of Proposed Object-Oriented Software Metrics*, Dissertation, School of Business, Virginia Commonwealth University, Richmond, VA, 1996.

13. Neal, R.D., R.J. Coppins, and H.R. Weistroffer, *The Assignment of Scale to Object-Oriented Software Measures*, Working Paper, Virginia Commonwealth University, Richmond, VA, 1997.

14. Neal, R.D., H.R. Weistroffer, and R.J. Coppins, *An Improved Suite of Object-Oriented Software Measures*, Working Paper, Virginia Commonwealth University, Richmond, VA, 1997.

15. Rumbaugh, James, Michael Blaha, William Premerlani, Frederick Eddy, and William Lorensen, *Object-Oriented Modeling and Design*, Prentice Hall, Englewood Cliffs, NJ, 1991.

16. Smith, David N., *Concepts of Object-Oriented Programming*, McGraw-Hill, NY, 1991.

17. Sully, Phil, *Modeling the World with Objects*, Prentice Hall, NY, 1993.

18. Tegarden, David P., Steven D. Sheetz, and David E. Monarchi, A Software Complexity Model of Object-Oriented Systems, *Decision Support Systems 13*, 1995, 241-62.

19. Zuse, Horst, *Software Complexity: Measures and Methods*, Walter de Gruyter, Berlin, 1990.

20. Zuse, Horst, "Foundations of Object-Oriented Software Measures", *Proceedings of the Third International Software Metrics Symposium*, March 1996.